Translation Validation of Decompilation

Sandeep Dasgupta
sdasgup3@illinois.edu

Advisor: Vikram S. Adve

University of Illinois at Urbana Champaign, USA

1 Abstract

The ability to directly reason about the binary code is desirable, not only because, in many circumstances, it is the only way to prove (or disprove) properties of the code that is actually executed (for example, when the source code is not available in case of legacy code or malware), but also because it avoids the need to trust the correctness of compilers.

Binary analysis is generally performed by existing decompiler projects [9, 5, 10, 7, 6], by (1) translating machine code to a intermediate representation (IR), and thereby exposing many high-level properties (like control flow, function boundary and prototype, variable and their type etc.) of the the binary, which are otherwise lost during the compilation pipeline, and (2) performing the analysis at the IR level. Analyzing the binary using the abstractions lifted to such high-level IR assist further analysis and/or optimization. Even though such analyzes are agnostic to any specific high-level IR, but many projects [9, 5, 1, 4, 3] prefer to employ LLVM IR. LLVM IR, being an industry standard compiler IR, enables many analyses and optimizations out-of-the-box which allows building a static binary analyzer with minimal effort.

Formally establishing faithfulness of the decompilation (i.e. translation from machine code to high level IR) is pivotal to gain trust in any binary analysis and, to the best of our knowledge, there is no exiting work to formally validate the translation of binary to LLVM IR.We believe an effort in the direction would benefit existing LLVM based binary analyzers.

The goal of our work is to formally validate such translation by leveraging the semantics of the languages involved (e.g. the Intel's X86-64 and LLVM IR). Some high level steps towards that goal involves defining the semantics of X86-64 and LLVM IR using K framework, a framework to define language semantics, and using the formal analysis tools which K provides "out-of-the-box" to develop a equivalence checker in order to prove equivalence between programs written in those languages. We have already taken the first big step towards the goal by formally defining the most complete X86-64 user-level ISA [8]. Moreover, the semantics of a subset of LLVM IR is already available [2] for us to leverage and build on.

Given the recent advances in translation validation in validating compilation, a basic version of this strategy is likely quite feasible today, however, there are additional challenges to deal with when it comes to validating the decompilation pipeline, like finding function, basic block and variable correspondence for checking equivalence, identifying and pruning glue code added during decompilation, to make the equivalence checker aware of the optimizations performed by the decompiler (e.g. recovering prototype and variables). We would like validation approach to be general enough and hence applicable to any state-of-the-art binary to LLVM IR translators [9, 5, 1, 4, 3], thereby avoiding any translator specific customization during validation. However, having this goal makes the problem even more challenging.

References

[1] *fcd: An optimizing decompiler.* `https://zneak.github.io/fcd/`. Last accessed: April 6, 2019.

[2] *Formal semantics of llvm ir in k.* `https://github.com/kframework/llvm-semantics`. Last accessed: April 6, 2019.

[3] *llvm-mctoll.* `https://github.com/Microsoft/llvm-mctoll`. Last accessed: April 6, 2019.

[4] *reopt: A tool for analyzing x86-64 binaries.* `https://github.com/GaloisInc/reopt`. Last accessed: April 6, 2019.

[5] *Remill: Library for lifting of x86, amd64, and aarch64 machine code to llvm bitcode.* `https://github.com/trailofbits/remill`, July 2018.

[6] S. Alvarez, *Radare2.* `https://rada.re/r/`, July 2018.

[7] D. Brumley, I. Jager, T. Avgerinos, and E. J. Schwartz, *Bap: A binary analysis platform*, in Proceedings of the 23rd International Conference on Computer Aided Verification, CAV'11, Berlin, Heidelberg, 2011, Springer-Verlag, pp. 463–469.

[8] S. Dasgupta and V. Adve, *A complete formal semantics of x86-64 user-level instruction set architecture*, in (Accepted, Publication Awaited) in Proceedings of the 2019 ACM SIGPLAN Conference on Programming Language Design and Implementation, ACM, 2019.

[9] A. Ruef and A. Dinaburg, *Static Translation of X86 Instruction Semantics to LLVM with McSema*, REcon, (2014).

[10] Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel, and G. Vigna, *Sok: (state of) the art of war: Offensive techniques in binary analysis*, (2016).